UNIFESP
UNIVERSIDADE FEDERAL DE SÃO PAULO
1933
Instituto de Ciência e Tecnologia
Universidade Federal de São Paulo

Compiladores:
Gramáticas livres de contexto e Árvores de análise sintática

Profª Thaína A. A. Tosta

tosta.thaina@unifesp.br

São José dos Campos – 2021/2

Gramáticas livres de contexto e Árvores de análise sintática

- A análise sintática determina a sintaxe ou estrutura de um programa a partir dos tokens reconhecidos pelo analisador léxico;

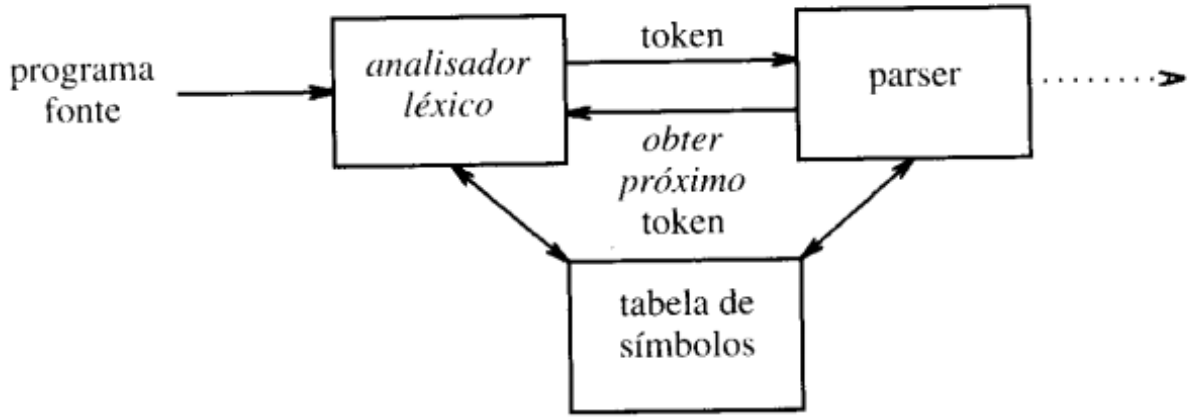


Fig. 3.1. Interação do analisador léxico com o *parser*.

- A sintaxe de uma linguagem de programação é dada pelas regras gramaticais de uma gramática livre de contexto, que tem convenções para nomes e operações similares às usadas por expressões regulares, mas com regras recursivas e possível aninhamento de declarações.

Gramáticas livres de contexto e Árvores de análise sintática

- A gramática livre de contexto é uma especificação para a estrutura sintática de uma linguagem de programação, similar à estrutura léxica por expressões regulares;
- A recursividade que caracteriza as gramáticas pode ser observada no exemplo para definição de expressões aritméticas de adição, subtração e multiplicação:

$$exp \rightarrow exp\ op\ exp \mid (\ exp\ ) \mid \textbf{número}$$
$$op \rightarrow + \mid - \mid *$$

- As regras gramaticais nessa forma são normalmente chamadas de forma de Backus-Naur, ou BNF.

Gramáticas livres de contexto e Árvores de análise sintática

- As regras gramaticais e as expressões regulares são definidas sobre um alfabeto ou um conjunto de símbolos;
- Nas expressões regulares, o alfabeto é formado pelos caracteres, mas as regras gramaticais utilizam como símbolos os tokens.

```
{if,then,else,end,repeat,until,read,write,
identificador,número,+,-,*,/,=,<,(,),;,:=}
```

$$exp \rightarrow exp\ op\ exp \mid (\ exp\ ) \mid \textbf{número}$$
$$op \rightarrow + \mid - \mid *$$

```
<exp> ::= <exp> <op> <exp> | (<exp>) | NÚMERO
<op> ::= + | - | *
```

Gramáticas livres de contexto e Árvores de análise sintática

- Pelas regras gramaticais, conseguimos determinar se uma cadeia de tokens é válida ou não.

  ✓(34-3)*42 × (34-3*42

- Para saber se uma cadeia de símbolos é ou não uma cadeia válida, as regras gramaticais utilizam as derivações, que são sequências de substituições pelas regras gramaticais;

Os nomes de estruturas são os não-terminais, pois eles podem ser substituídos na derivação

$$exp \rightarrow exp\ op\ exp \mid (\ exp\ ) \mid \textbf{número}$$
$$op \rightarrow + \mid - \mid *$$

(1)	$exp \Rightarrow exp\ op\ exp$	$[exp \rightarrow exp\ op\ exp]$
(2)	$\Rightarrow exp\ op$ **número**	[exp → **número**]
(3)	$\Rightarrow exp$ * **número**	[op → *]
(4)	⇒ (exp) * **número**	[exp → (exp)]
(5)	⇒ (exp op exp) * **número**	$[exp \rightarrow exp\ op\ exp]$
(6)	⇒ (exp op **número**) * **número**	[exp → **número**]
(7)	⇒ (exp - **número**) * **número**	[op → -]
(8)	⇒ (**número** - **número**) * **número**	[exp → **número**]

Figura 3.1 Uma derivação para a expressão aritmética (34-3)*42.

Os símbolos do alfabeto são denominados terminais, porque eles terminam a derivação.

Gramáticas livres de contexto e Árvores de análise sintática

A gramática para uma linguagem de programação comumente define uma estrutura denominada programa, e o conjunto de cadeias de símbolos dessa linguagem é o conjunto de símbolos sintaticamente válidos.

programa → *programa-cabeçalho* **;** *programa-bloco* **.**
programa-cabeçalho → . . .
programa-bloco → . . .
. . .

Gramáticas livres de contexto e Árvores de análise sintática

Considere a gramática de declarações extremamente simplificada a seguir, que pode ser escrita usando uma ε-produção:

declaração → *if-decl* | **outra**
if-decl → **if** (*exp*) *declaração*
| **if** (*exp*) *declaração* **else** *declaração*
exp → **0** | **1**

declaração → *if-decl* | **outra**
if-decl → **if** (*exp*) *declaração else-parte*
else-parte → **else** *declaração* | ε
exp → **0** | **1**

```
outra
if (0) outra
if (1) outra
if (0) outra else outra
if (1) outra else outra
if (0) if (0) outra
if (0) if (1) outra else outra
if (1) outra else if (0) outra else outra
...
```

Gramáticas livres de contexto e Árvores de análise sintática

- Uma derivação permite a construção de uma cadeia específica de terminais a partir de um não-terminal inicial;
- Em geral, existem muitas derivações para a mesma cadeia;
- Precisamos de uma estrutura que não represente essas diferenças superficiais, as árvores de análise sintática.

(1) *exp*	⇒ *exp op exp*	[*exp* → *exp op exp*]
(2)	⇒ *exp op* **número**	[*exp* → **número**]
(3)	⇒ *exp* * **número**	[*op* → *]
(4)	⇒ (*exp*) * **número**	[*exp* → (*exp*)]
(5)	⇒ (*exp op exp*) * **número**	[*exp* → *exp op exp*]
(6)	⇒ (*exp op* **número**) * **número**	[*exp* → **número**]
(7)	⇒ (*exp* - **número**) * **número**	[*op* → -]
(8)	⇒ (**número** - **número**) * **número**	[*exp* → **número**]

Figura 3.1 Uma derivação para a expressão aritmética `(34-3)*42`.

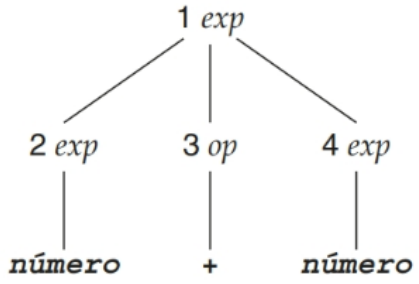
(1) *exp*	⇒ *exp op exp*	[exp → *exp op exp*]
(2)	⇒ (*exp*) *op exp*	[exp → (*exp*)]
(3)	⇒ (*exp op exp*) *op exp*	[*exp* → *exp op exp*]
(4)	⇒ (**número** *op exp*) *op exp*	[*exp* → **número**]
(5)	⇒ (**número** - *exp*) *op exp*	[*op* → -]
(6)	⇒ (**número** - **número**) *op exp*	[*exp* → **número**]
(7)	⇒ (**número** - **número**) * *exp*	[*op* → *]
(8)	⇒ (**número** - **número**) * **número**	[*exp* → **número**]

Figura 3.2 Outra derivação para a expressão `(34-3)*42`.

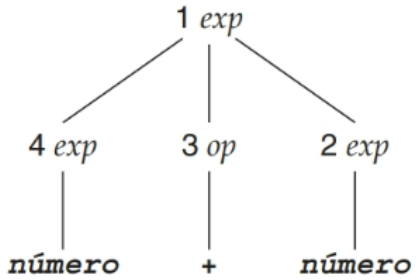
Gramáticas livres de contexto e Árvores de análise sintática

Uma árvore de análise sintática correspondente a uma derivação é uma árvore com nós interiores rotulados por não-terminais, nós-folha rotulados por terminais e os filhos de cada nó interno representam a substituição do não-terminal em um passo da derivação.

(1) *exp* ⇒ *exp op exp*
(2) ⇒ **número** *op exp*
(3) ⇒ **número + *exp***
(4) ⇒ **número + número**



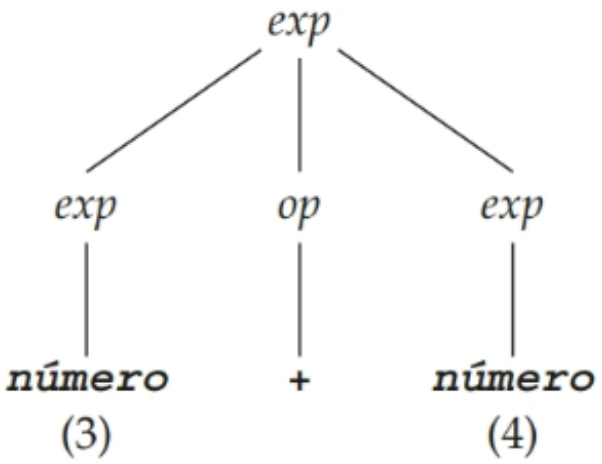
exp ⇒ *exp op exp*
⇒ *exp op* **número**
⇒ *exp* **+ número**
⇒ **número + número**

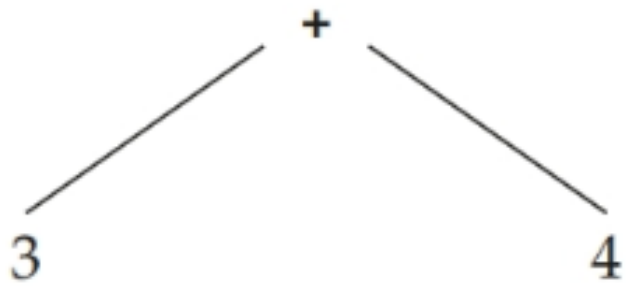


Derivação à esquerda com enumeração em pré-ordem	Derivação à direita com enumeração inversa em pós-ordem

Gramáticas livres de contexto e Árvores de análise sintática

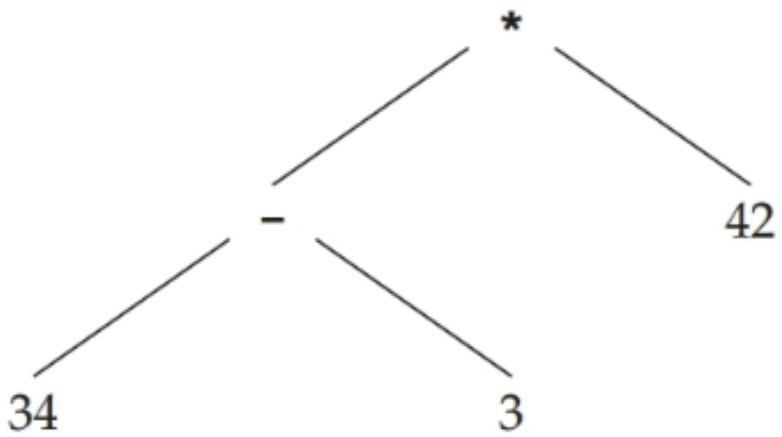
- Uma árvore de análise sintática contém muito mais informação que o que é necessário para o compilador;
- Considere a árvore para representação da expressão 3+4, cuja simplificação é definida pelo **princípio da tradução direcionada por sintaxe** que estabelece que o significado da cadeia 3+4 deveria ser diretamente relacionado a sua estrutura sintática representada na árvore.





Gramáticas livres de contexto e Árvores de análise sintática

- As representações simplificadas das árvores são chamadas de árvores abstratas de análise sintática, ou árvores sintáticas abstratas;
- Um analisador sintático efetua todos os passos representados na árvore de análise sintática, mas em geral constrói só a árvore abstrata.



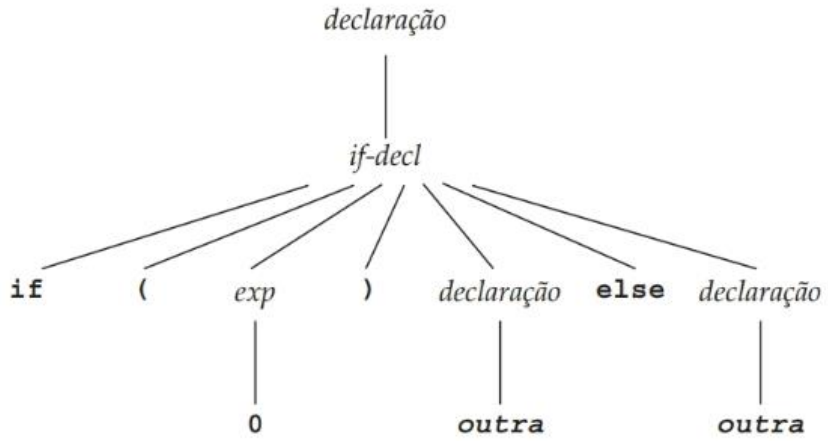
(34-3)*42

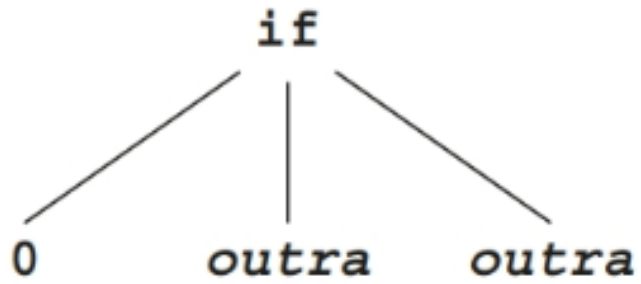
Gramáticas livres de contexto e Árvores de análise sintática

- Considere a gramática para declarações if simplificadas:

declaração → *if-decl* | **`outra`**
if-decl → **`if (`** *exp* **`)`** *declaração*
| **`if (`** *exp* **`)`** *declaração* **`else`** *declaração*
exp → **`0`** | **`1`**

- A árvore de análise sintática para a cadeia **`if (0) outra else outra`** é definida abaixo, com sua árvore abstrata:

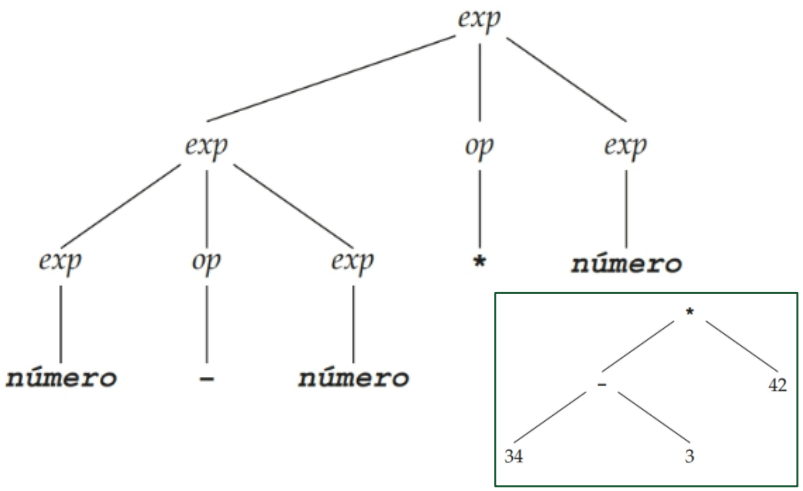


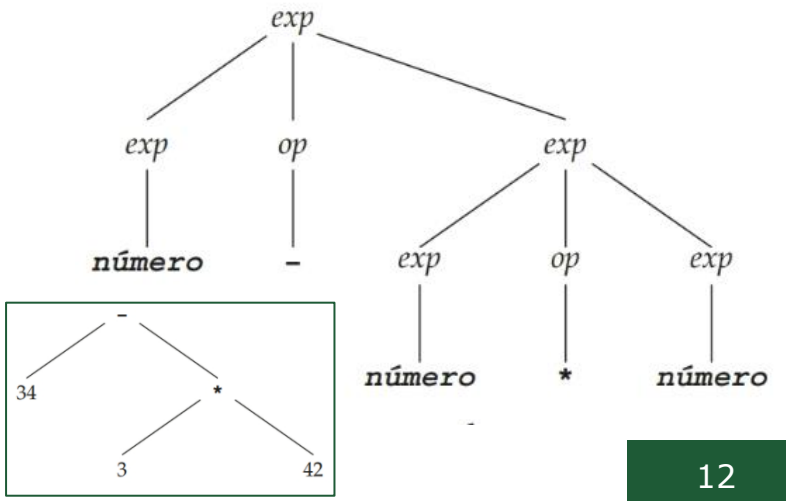


Gramáticas livres de contexto e Árvores de análise sintática

- Uma gramática pode permitir que uma cadeia tenha mais de uma árvore de análise sintática, sendo chamada de **gramática ambígua**;
- Para exemplificar essa situação, vamos utilizar a gramática de aritmética de inteiros e a cadeia **`34-3*42`**:

$$exp \rightarrow exp\ op\ exp \mid (\ exp\ ) \mid \textbf{número}$$
$$op \rightarrow + \mid - \mid *$$





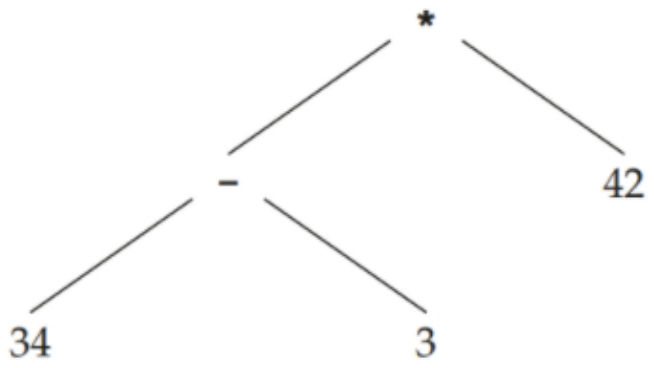
Gramáticas livres de contexto e Árvores de análise sintática

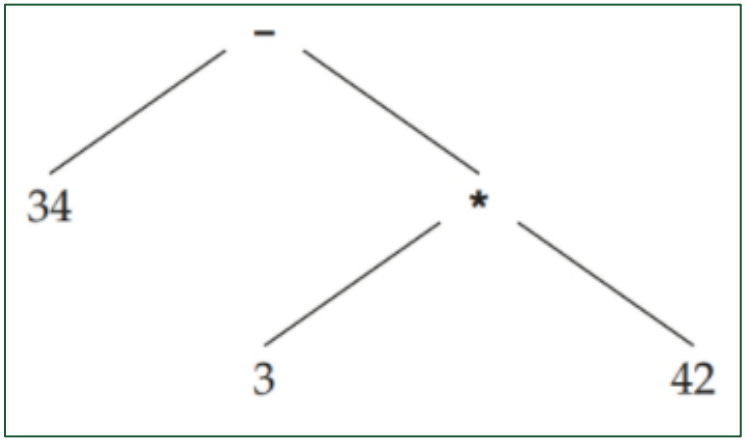
- As ambiguidades podem ser tratadas por dois métodos básicos:
  - Definição de uma regra de eliminação de ambiguidade: a gramática não é alterada mas a estrutura sintática da linguagem passa a ser definida pela gramática e essa regra;
  - Alteração da gramática para forçar a construção da árvore correta.
- Independentemente do método que utilizamos, primeiro decidimos qual árvore em um caso ambíguo é a correta, retomando o princípio da tradução direcionada pela sintaxe.

Gramáticas livres de contexto e Árvores de análise sintática

O conceito de precedência pode ser levado em consideração na formulação da gramática, o que resolve essa ambiguidade.

34-3*42





Gramáticas livres de contexto e Árvores de análise sintática

Para incorporar o conceito de precedência na gramática, podemos agrupar as operações com a mesma precedência e criar uma regra específica para cada uma delas.

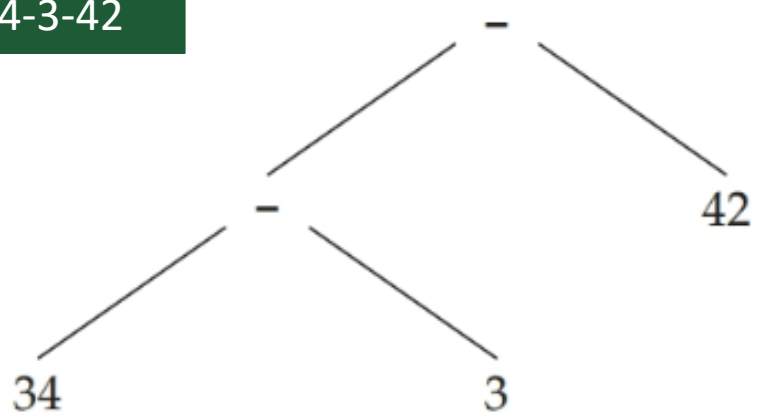
$$exp \rightarrow exp\ op\ exp \mid (\ \ exp\ ) \mid \textbf{número}$$
$$op \rightarrow + \mid - \mid *$$

$$exp \rightarrow exp\ soma\ exp \mid termo$$
$$soma \rightarrow + \mid -$$
$$termo \rightarrow termo\ mult\ termo \mid fator$$
$$mult \rightarrow *$$
$$fator \rightarrow (\ \ exp\ ) \mid \textbf{número}$$

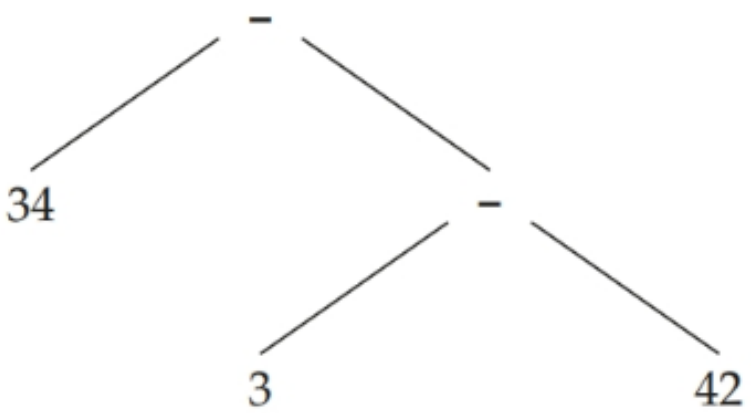
Gramáticas livres de contexto e Árvores de análise sintática

- Uma segunda ambiguidade possível na gramática de operações aritméticas é a ordem de aplicação de operadores de mesma precedência;
- É necessário definir ainda a associatividade dos operadores.

34-3-42



Associatividade à esquerda



Associatividade à direita

Gramáticas livres de contexto e Árvores de análise sintática

- A gramática que definimos para lidar com as precedências poderia ser alterada para considerar a associatividade à esquerda;
- Para fazer isso, temos que substituir a regra **`exp`** por **`termo`**, limitando possíveis repetições apenas ao lado esquerdo.

$exp \rightarrow exp\ soma\ \boxed{exp} \mid termo$
$soma \rightarrow \texttt{+} \mid \texttt{-}$
$termo \rightarrow termo\ mult\ termo \mid fator$
$mult \rightarrow \texttt{*}$
$fator \rightarrow \texttt{(}\ \ exp\ \texttt{)} \mid \texttt{número}$

$exp \rightarrow exp\ soma\ \boxed{termo} \mid termo$
$soma \rightarrow \texttt{+} \mid \texttt{-}$
$termo \rightarrow termo\ mult\ fator \mid fator$
$mult \rightarrow \texttt{*}$
$fator \rightarrow \texttt{(}\ \ exp\ \texttt{)} \mid \texttt{número}$

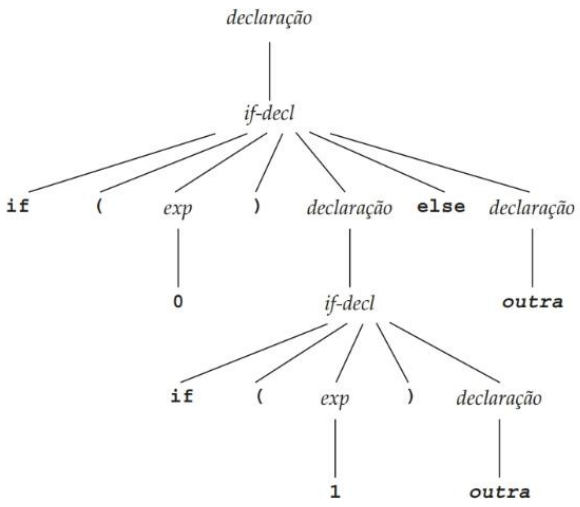
Gramáticas livres de contexto e Árvores de análise sintática

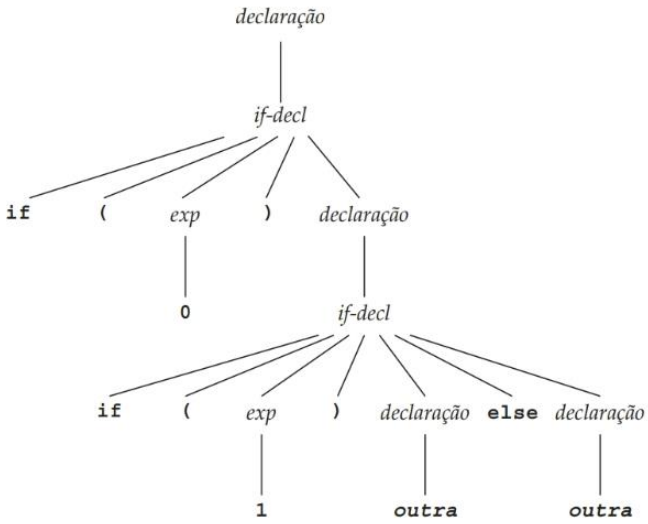
A gramática abaixo é ambígua em decorrência do problema do else opcional:

declaração → *if-decl* | **outra**
if-decl → **if** **(** *exp* **)** *declaração*
| **if** **(** *exp* **)** *declaração* **else** *declaração*
exp → **0** | **1**

---

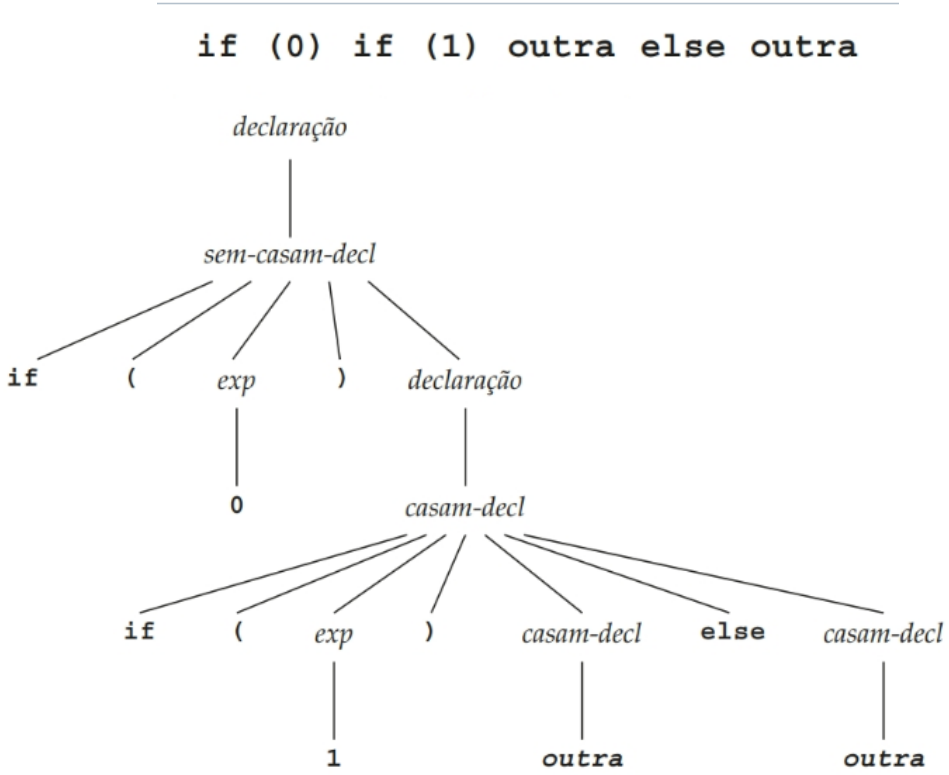
if (0) if (1) outra else outra





Gramáticas livres de contexto e Árvores de análise sintática

- Uma regra para eliminação desse tipo de ambiguidade é pela regra do aninhamento mais próximo;
- Para resolver esse problema, poderíamos atualizar a gramática com dois tipos de regras: com **`casam-decl`** e **`sem-casam-decl`**.



declaração → *casam-decl* | *sem-casam-decl*
casam-decl → **`if`** **`(`** *exp* **`)`** *casam-decl* **`else`** *casam-decl* | **`outra`**
sem-casam-decl → **`if`** **`(`** *exp* **`)`** *declaração*
| **`if`** **`(`** *exp* **`)`** *casam-decl* **`else`** *sem-casam-decl*
exp → **`0`** | **`1`**

Gramáticas livres de contexto e Árvores de análise sintática

Uma segunda opção é adicionar a palavra-chave **`end if`** para marcar o fim do bloco **`if`**, compondo uma solução pelo uso de uma palavra-chave de marcação de bloco para essa declaração.

if-decl → **`if`** *condição* **`then`** *seq-decl* **`end if`**
| **`if`** *condição* **`then`** *seq-decl*
`else` *seq-decl* **`end if`**

```
if x /= 0 then
   if y = 1/x then ok := true;
   else z := 1/x;
   end if;
end if;
```

```
if x /= 0 then
   if y = 1/x then ok := true;
   end if;
else z := 1/x;
end if;
```

Gramáticas livres de contexto e Árvores de análise sintática

Construções repetitivas podem ter notações especiais utilizando uma notação BNF estendida, que é chamada de EBNF.

$A \rightarrow A\ \alpha \mid \beta$ (recursiva à esquerda) $A \rightarrow \beta\ \alpha^*$ $A \rightarrow \beta\{\alpha\}$

$A \rightarrow \alpha\ A \mid \beta$ (recursiva à direita) $A \rightarrow \alpha^*\ \beta$ $A \rightarrow \{\alpha\}\beta$

Gramáticas livres de contexto e Árvores de análise sintática

decl-seqüência → *decl* **;** *decl-seqüência* | *decl*
decl → **s**

$A \rightarrow \alpha\, A \mid \beta$, em que A = *decl-seqüência*, α = *decl* **;** e β = *decl*.

$A \rightarrow \{\alpha\}\beta$ *decl-seqüência* → { *decl* **;** } *decl*

exp → *exp soma termo* | *termo*

$A \rightarrow A\, \alpha \mid \beta$, em que A = *exp*, α = *soma termo* e β = *termo*.

exp → *termo* { *soma termo* }

Gramáticas livres de contexto e Árvores de análise sintática

Construções opcionais em EBNF são indicadas por colchetes:

declaração → *if-decl* | **`outra`**
if-decl → **`if ( `** *exp* **`)`** *declaração*[**`else`** *declaração*]
exp → **`0`** | **`1`**

decl-seqüência → *decl* **`;`** *decl-seqüência* | *decl*

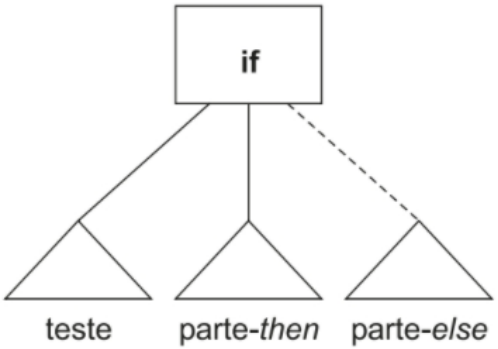
decl-seqüência → *decl* [**`;`** *decl-seqüência*]

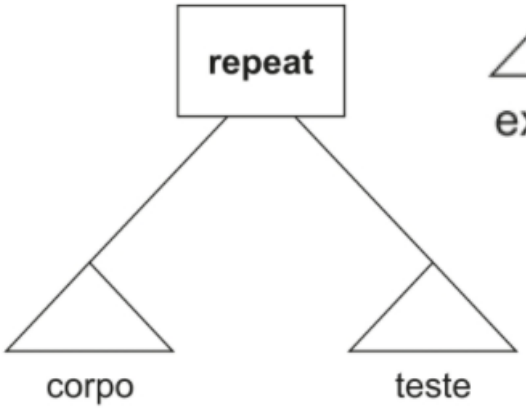
exp → *termo soma exp* | *termo*

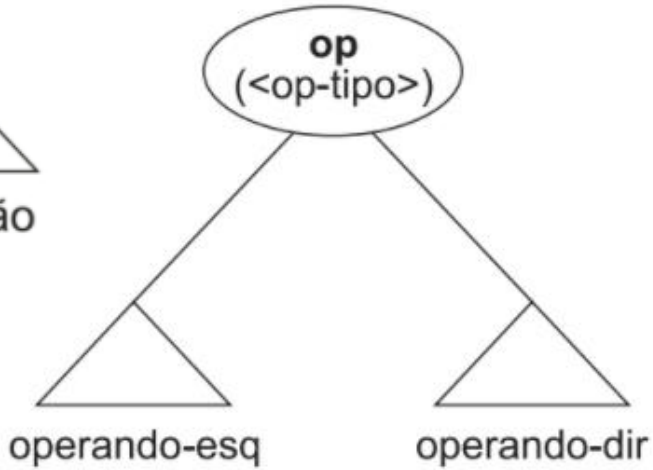
exp → *termo* [*soma exp*]

Gramáticas livres de contexto e Árvores de análise sintática

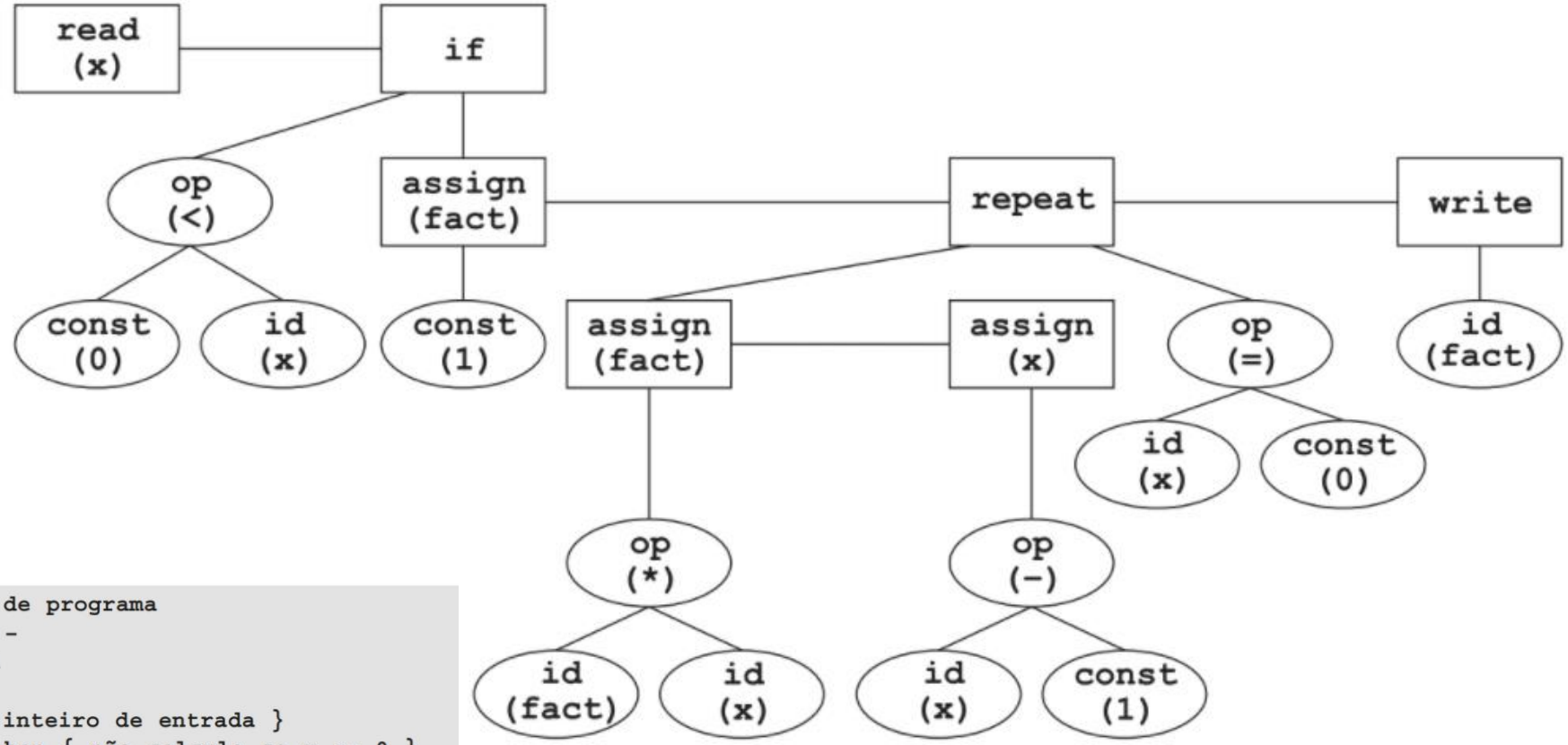
Para uma descrição visual da árvore sintática, podemos utilizar as seguintes definições:







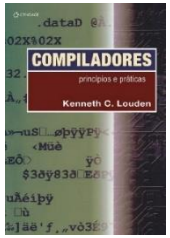
Gramáticas livres de contexto e Árvores de análise sintática




```
{ Exemplo de programa
  em TINY -
  fatorial
}
read x; { inteiro de entrada }
if 0 < x then { não calcula se x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact { apresenta o fatorial de x }
end
```

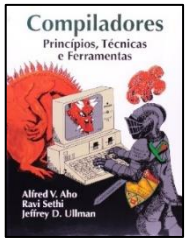
Gramáticas livres de contexto e Árvores de análise sintática

Bibliografia consultada



LOUDEN, K. C. **Compiladores: princípios e práticas.** São Paulo: Pioneira Thompson Learning, 2004.

RICARTE, I. **Introdução à Compilação.** Rio de Janeiro: Editora Campus/Elsevier, 2008.



AHO, A. V.; LAM, M. S.; SETHI, R. e ULLMAN, J. D. **Compiladores: princípios, técnicas e ferramentas.** 2ª edição – São Paulo: Pearson Addison-Wesley, 2008.

Agradeço ao Professor Luiz Eduardo G. Martins pelo material disponibilizado.